Technical Disclosure Commons

Defensive Publications Series

November 2019

Automatic Design of Neural Architectures for Recommendation and Ranking Tasks

Anonymous

Automatic Design of Neural Architectures for Recommendation and Ranking Tasks

ABSTRACT

Designing the right neural network architecture for a given machine-learning task is critical for performance. For example, the most appropriate neural networks for tasks such as image classification, speech recognition, click-through-rate prediction, etc. are different from each other. This disclosure describes a framework for conducting searches for neural architectures that perform recommendation and ranking tasks.

KEYWORDS

- Neural network (NN)
- Neural architecture
- Automatic neural network design
- Multi-layer perceptron
- Generalized cross-net
- Factorization machine
- Dot processor

BACKGROUND

Designing the right neural network architecture for a given machine-learning task is critical for performance. For example, the most appropriate neural networks for tasks such as image classification, speech recognition, click-through-rate prediction, etc. are different from each other.

Prior work in this domain includes techniques to compose a neural network using components, e.g., as used in computer vision or speech applications. Another approach is to define the architecture search with a directed acyclic graph (DAG).

DESCRIPTION

It a challenge to define the search space for deep recommendation models since the model architectures are more heterogeneous than that of computer vision (CV) models. This disclosure describes a framework for conducting searches for deep, heterogeneous neural architectures that perform recommendation and ranking tasks.

The techniques described herein find common building blocks from popular models and search for different ways in which they can be coupled. The modules (building blocks) can include, e.g., Cross-Net, DotProcessor, factorization machine (FM), multi-layer perceptron (MLP), CatProcessor, etc. The modules or building blocks are selected to satisfy the following criteria.

- **Homogeneity:** The final performance should be continuous with respect to the module hyperparameters. The module hyperparameters include, for example in the case of a multi-layer perceptron (MLP), the number of layers, layer width, dropout probability, etc.
- **Functionality:** The functionality of the module should be appropriate with the function of the neural network. For example, for feature interaction, dot processor is appropriate. The hypothesis can also be checked quantitatively to provide an additional signal for performance evaluation.
- **Cost-awareness:** The computational, e.g., time and memory, costs of a module should be a simple function of its input-output specifications and of its hyperparameters.

Table 1 illustrates example modules that constitute a neural network, along with their parameters, functionalities, and time-space cost.

Building block	**Building-block hyper-parameters to be searched**	**Functionality**	**Memory cost in time or space (number of parameters)**
Multi-layer perceptron (MLP)	Input feature IDs (input dimension:) d_{in} Layer width: O Output dimension: d_{out}	Basic feature translation	Time: $O(N \times d_{in} \times d_{out})$ Space: $d_{in} \times d_{out} + d_{out}$
Generalized cross-net (GCN)	Two sets of input feature IDs, with their concatenation vectors converted to the same dimension (as necessary) using linear transformation. Input dimension: d_{in}^A, d_{in}^B Layer output width: O Output dimension: d_{out}	Skip connection with cross product	Time: $O(N \times (d_{in}^A + d_{in}^B) \times d_{out})$ Space: $(d_{in}^A + d_{in}^B + 2) \times d_{out}$
Dot processor	Input feature IDs (Input feature number: f Input feature embeddings dimension: d) Output dimension: f(f-1)/2	Pairwise dot product	Time: $O(N \times d \times f^2)$ Space: 0
Factorization machine (FM)	Input feature IDs (Input feature number: f Input feature embeddings dimension: d) Output dimension: 1	Factorization machine (sum of dot processor) + first-order embedding	Time: $O(N \times f \times d \times 3)$ Space: 0
CatProcessor		Assistant function	

Table 1: Example modules that can be used to construct a neural network, along with their properties

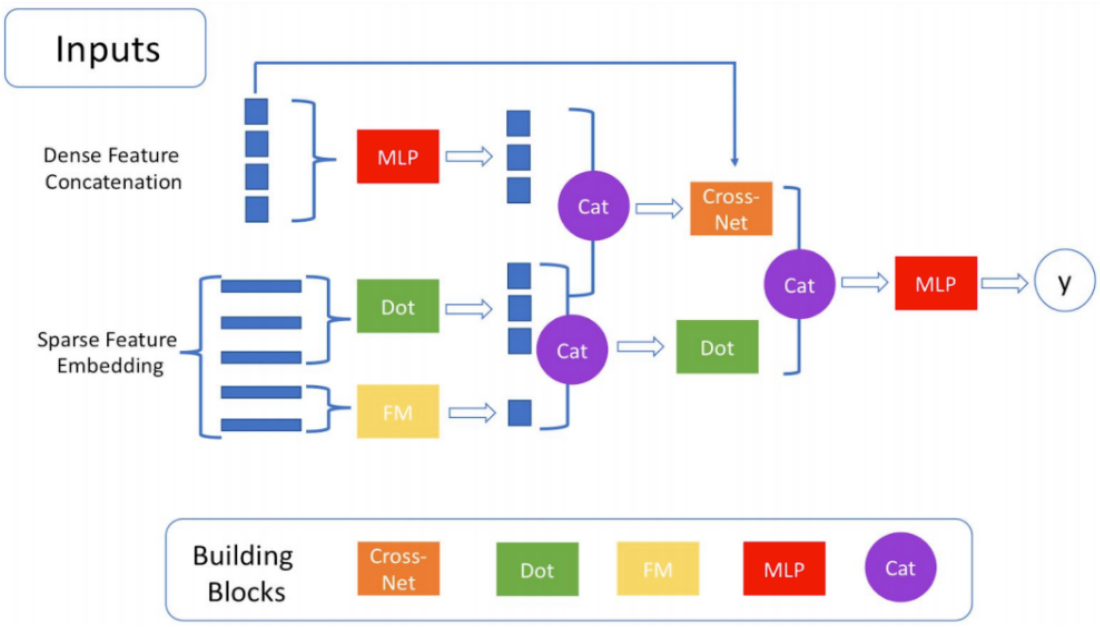

Fig. 1: Constructing a neural network architecture from modules

Fig. 1 illustrates constructing a neural network architecture from modules (building blocks), per techniques of this disclosure. The neural network is constructed as a directed acyclic graph (DAG) over a number of modules such as Cross-Net, DotProcessor, factorization machine (FM), multi-layer perceptron (MLP), cat processor, etc.

Connections between blocks

Connections between the blocks, or nodes of the DAG, are added in stages. At a given stage, the added blocks consume input from outputs of a subset of the existing nodes as well as raw inputs. Size mismatch is fixed by a single, fully-connected layer.

There are two types of output, e.g., dense or embeddings. Sparse features are converted to embeddings by default, e.g., they share embeddings for different modules. Dense features or outputs are concatenated by default. Embedding features or outputs are treated separately. A type-qualifier to the embeddings output enables such use cases as applying the dot processor on

the outputs of several modules. Alternatively, the output type can be limited to one type, e.g., dense.

The neural network architecture can be described as an element in a search space that comprises DAGs over a relatively small number of modules. The techniques of this disclosure enable efficient coverage of the search space. A module can be broken into smaller pieces, and the DAG enlarged, to enable the search of a larger space. Constraints such as the upper bounds on block numbers and layer widths are predefined, and, to avoid intermediate data explosions, follow the computational, e.g., RAM, limitations.

Searching for an optimal neural network architecture

Per techniques of this disclosure, the search strategy is feature-aware, e.g., when choosing the inputs of a node, both the properties of the input and the functionality of the node are considered. For example, a dot processor should be able to create interactions between user-side features and object-side features. To achieve this, the input and the intermediate input/output features are characterized as follows.

Input features can be labeled directly using expert knowledge. For example, for a ranking and recommendation neural network, input labels such as “feature id,” “user-side,” “ads-side,” “historical counts,” “content feature,” etc. can be provided. This information is exposed to the searcher. The type of the intermediate input/output features can be inferred based on the input and the functionality of the module. For example, if a dot processor takes both user and ads-side features, its output can be labeled as “user-ads interaction.” Meta-information that is missing from publicly available data-sets can be simulated by creating pseudo meta-information, e.g., via

feature clustering. The search strategy is also cost-aware, e.g., the identified neural network candidate follows the limitations of computational complexity.

Example: Reinforcement-learning based search with RNN controller

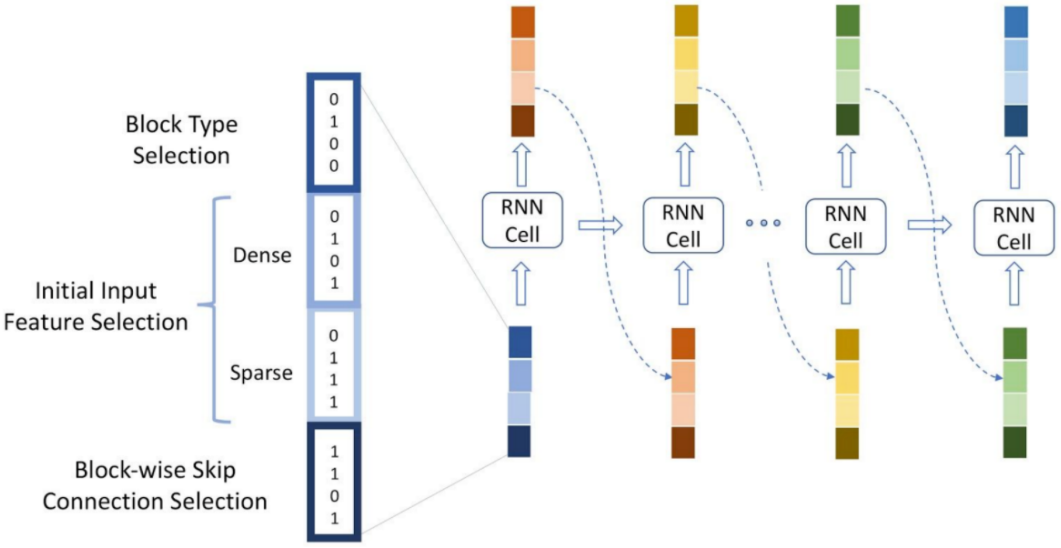

Fig. 2: Searching for an optimal RNN neural network architecture

Fig. 2 illustrates reinforcement-learning (RL) based searching with a recurrent neural network (RNN) architecture, per techniques of this disclosure. The RNN is represented in one of its topological orderings. The state of each node is represented using the following parameters.

- Node type: the type of module for the node and module hyperparameters.
- Inputs, including a subset of input features and previous module outputs, and the procedure to use them. A multi-hot vector can be used to indicate an input feature.
- Outputs, including output dimensions and characteristics, and overall complexity.

The search stops when the number of nodes reaches and/or complexity hit a limit. The final output is a simple logistic regression (LR) over a subset (or all) intermediate outputs. Some advantages of techniques of this disclosure are as follows:

1. Search is performed on a small dataset in a simple search space, and once an optimal architecture is found, it is merged into an existing architecture.
2. Joint search architecture and feature spaces.
3. The search space is split by dense or sparse features.
4. The techniques are cost-aware, e.g., they fit pre-defined limits on computational complexity.
5. The techniques decouple the searcher from the searched space.
6. The techniques apply an evolutionary procedure on a directed acyclic graph (DAG) based search space.
7. An optimal architecture is arrived at by gradually adding components

CONCLUSION

This disclosure describes a framework for conducting searches for neural architectures that perform recommendation and ranking tasks.

REFERENCES

[1] Real, Esteban, Alok Aggarwal, Yanping Huang, and Quoc V. Le. “Regularized evolution for image classifier architecture search.” *arXiv preprint* arXiv:1802.01548 (2019).

[2] Wang, Linnan, Saining Xie, Teng Li, Rodrigo Fonseca, and Yuandong Tian. "Sample-Efficient Neural Architecture Search by Learning Action Space." *arXiv preprint* arXiv:1906.06832 (2019)